Texas A&M University - Corpus Christi Genomics Core Lab Metabarcoding Manual

Evan Krell, Chris Bird, Martin French, Sharon Magnuson

ABSTRACT

1 INTRODUCTION

The GCL metabarcoding process involves numerous scripts in a variety of programming languages. A metabarcoding project is done by creating a pipeline of these scripts and setting the parameters of each. Because the needs of individual projects differs, it is difficult to create a one-size-fits-all program. Therefore, the use of small, modular programs appears to be the most effective. This manual describes each script in the GCL pipeline, demonstrates an example pipeline, and offers guidance on setting up databases and running on an HPC.

Multiple assignment methods are supported and my be run simultaneously. For example, a user can supply options to use both BLAST and VSEARCH. At the assignment step, the pipeline splits and both methods are run in parallel. Each method produces a number of resulting files. These files are differentiated by including the assignment method used in the path name.

Taxonomic assignment is not required. If no assignment databases are specified, the pipeline can still be used for just the filtering and clustering of reads.

1.1 Charybdis

All of the code resides in a repository called `GCL_Charybdis`. The repository is located on the HPC at `/work/hobi/GCL/GCL_Charybdis`. The name resulted from finding a critter from species *Charybdis* from our first set of metabarcoding results.

2 PIPELINE PARAMETERS

You need a number of files and bits of information in order to run the pipeline. The following describes each parameter, and (if relevant), how to obtain them. See section 4 for actual examples of these parameters used in a project.

2.0.1 Project Name (-p)

The name of the project is used as a prefix to many files generated throughout the pipeline. Additionally, the input files need to use this project name in their filenames.

2.0.2 Input Directory (-i)

Certain files need to be placed in this directory before executing the pipeline. Instead of forcing the user to specify several files, they are placed in a single location. The pipeline finds files by using the project name specified with the `-p` option. The required files for running the generic pipeline script is shown in Table 1.

2.0.3 Output Directory (-o)

Files generated by pipeline will be placed in here.

2.0.4 Chunks (-n)

Number of parallel threads to spawn. Use all the cores available if running on an HPC node. But if running on desktop, use less than your number of available cores. For the 40-core workstation in Bird's computer room, I would go with 35.

Table 1. Files required to exist in the input directory.
The variable project name is represented by `<projectname>`.

`<projectname>`_forward.fastq	FASTQ with forward reads
`<projectname>`_reverse.fastq	FASTQ with forward reads
`<projectname>`.barcodes.txt	Tab-delimited file to match up barcodes to samples
`<projectname>`.sampledescs.csv	CSV that matches descriptive names to samples

2.0.5 BLAST Database (-b)

Any valid BLAST database. GCL typically uses the the nucleotide (nt) database from NCBI.
`ftp://ftp.ncbi.nlm.nih.gov/blast/db/`

2.0.6 BLAST Ignore File (-d)

A text file containing GIs of sequences in the BLAST database to ignore. It is fine to have GIs that are not actually in the BLAST database used. Good for filtering environmental DNA and such.

2.0.7 VSEARCH Database (-v)

VSEARCH is an alternative taxonomic assignment program Rognes et al. (2016). A valid VSEARCH database is a FASTA file. But because FASTA files are a loose standard, not all are compatible. VSEARCH gets unhappy regarding certain formats of the sequence header. An example of a valid FASTA entry is as follows (data is from (Ratnasingham and Hebert, 2007). Note that the `|` symbol is used to separate metadata field in the header. The first field is a unique sequence ID. The second is a scientific name. The third is the genetic region of the sequence. The fourth is the Barcode of Life Database (BOLD) sequence ID. It could be a NCBI GI or Accession instead.

```
>GBMAA467X14|Pomphorhynchus_laevis|COIX5P|KF559289
ATGTATGTTTTGGTTGGTGTGTGAGGGGGGCTAATGGGGTTTTCTATAAGACTATTAATTCGA
```

The script `/work/hobi/GCL/GCL_Charybdis/bin/vsearch_getTAXIDfromBOLDseqid.sh` will get the NCBI taxonomic ID from a BOLD sequence ID.

2.0.8 Chimera Database (-c)

FASTA database used for detection on chimeric reads. Chimera detection can be done using a database or using the *de novo* method Rognes et al. (2016). Currently the pipeline is hard-coded to use the *de novo* method, but still requires a chimera database that will not be used. This issue is described in section 5.3.

2.0.9 Target Sequence Length (-x)

The target sequence length is the length of a read expected using PCR. The forward and reverse primers extract a portion of DNA between them. The value of this option is based on the primers you used, so check their documentation or ask whoever did the PCR.

2.0.10 Charybdis Scripts Directory (-g)

Instead of setting the environment `$PATH` variable or placing the scripts in a systemwide bin directory, the directory holding all the pipeline scripts is an option. Less conventional than `$PATH`-based methods, but very easy to deal with.

2.0.11 NCBI Taxonomy Database (-t)

This database is used to get an organism's scientific name from a numeric taxonomic ID. Taxonomy database downloaded from `ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/`.
It is the same data as `https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi`

3 PIPELINE ARCHITECTURE

Pipelines are bash scripts that just string together a group of other scripts/programs to perform the metabarcoding process. Example scripts are included in `/work/hobi/GCL/GCL_Charybdis/pipelines`. The script called `charybdis_generic` is expected to handle most projects with little customization.

The section will describe all of the steps of the `charybdis_generic` pipeline. Hopefully others will be able to use this information to create a pipeline tailored to other metabarcoding projects.

Throughout the text, *attribute* refers to a piece of metadata included in a FASTA sequence header. These are in the form of key-value pairs. Many tools in the OBITOOLS package create and modify attributes, so we made many of our own scripts follow this convention.

3.1 Merge Reads

Script: `/work/hobi/GCL/GCL_Charybdis/bin/mergeReads.sh`
Slurm wrapper: `/work/hobi/GCL/GCL_Charybdis/bin/mergeReads.slurm`

The forward and reverse reads and merged into a single FASTA file, and basic quality filtering is done. The reads are matched to sample ID which are included as metadata in the FASTA sequence headers.

3.1.1 Split FASTQ files

In order to run the next sections in parallel, the forward and reverse read FASTQ files are split into equally sized chunks using *fastq-splitter.pl*.

3.1.2 Join forward and reverse reads

After this point, the pipeline uses a single file of merged forward and reverse reads. This is done using OBITOOL's *illuminapairedend* tool. We have it reject reads whose alignment score score is under a threshold of 40.

3.1.3 Convert FASTQ to FASTA

Convert the merged files to FASTA. This is done because many tools operate only on FASTA and because we don't need the sequencer quality information after this point. We can always use the unique sequence ID to access it from the origin FASTQ, if needed.

3.1.4 Remove unaligned sequences

We use OBITOOL's *obigrep* to select only those sequences whose alignment mode is ”joined”. *Obigrep* can search for patterns of values of specified attributes.

3.1.5 Remove sequences whose alignment length is below threshold

We use OBITOOL's *obigrep* to select only those sequences whose alignment length is $\geq$ a threshold. We have this threshold set to 20.

3.1.6 Match sequences to sample ID based on barcode

OBITOOL's *ngsfilter* is used to attach an attribute called *sample* to each sequence. The match is allowed to have 2 sequence errors by setting the flag `-e 2`.

3.1.7 Concatenate into single FASTA

The small chunks are concatenated into a single FASTA file.

3.2 Filter Reads

Script: `/work/hobi/GCL/GCL_Charybdis/bin/ObiToolsPipeline.sh`
Slurm wrapper: `/work/hobi/GCL/GCL_Charybdis/bin/filterReads.slurm`

This was the first script written. At the time, the size of the project was not known and this file was intended to be the entire pipeline. Thus, the script is called `ObiToolsPipeline.sh` instead of the more appropriate `filterReads.sh`.

3.2.1 Split into smaller files by sample ID

As before, we form a smaller number of files for parallel processing. However, the sample ID is used to split files instead of an arbitrary size. The sample ID is added to the FASTA filename to keep track.

3.2.2 Remove unneeded attributes

OBITOOLS added a huge number of attributes in a previous step. We get rid of the irrelevant ones with `obiannotate`. Note that attributes are never truly discarded since the sequence ID can always be used to query the intermediate FASTAs generated at each step of the pipeline.

3.2.3 Keep only unique sequences

In order to lower the filesize (often substantially), duplicated sequences are merged into a single representative sequence. The *obimerge* attribute is used to record how many sequences are represented by that sequence. This is done using `obiuniq`.

3.2.4 Remove PCR errors

PCR errors are filtered using `obiclean`. After numerous experiments, we found that the following options were performing well for a set of COI sequences from Gulf of Mexico fish. Results from projects where we reused these settings have appeared to be fine.

Table 2. Settings for obiclean

-r	0.5	Controls when less abundant sequences are labeled as variants of a similar more abundant
-d	1	Max number of differences to be considered variants
-H		Keep only a representative of a set of variants.

3.2.5 Remove low read length

The amplified sequence should be a certain size. Using our COI primers, we expect the size to be 313 base pairs. However, the actual size is often a bit lower or higher. You supply your base pairs length as a pipeline argument. The allowed range is that size $\pm$ 15.

3.2.6 Remove chimeras

VSEARCH is used to remove chimeras using the *de novo* method.

3.2.7 Concatenate into single FASTA

The small chunks are concatenated into a single FASTA file.

3.3 Collapse into OTUs

Script: `/work/hobi/GCL/GCL_Charybdis/bin/ClusterOTU.sh`
Slurm wrapper: `/work/hobi/GCL/GCL_Charybdis/bin/ClusterOTU.slurm`

3.3.1 Cluster with CROP

CROP clusters sequences into OTUs (Hao et al., 2011). CROP has a number of parameters that control the OTU assignment. Assigning these parameters is tricky, but the CROP manual has a recommended process for determining the values (TingChenLab, 2017). We initially used the recommendation, but then modified it to get better results. However, we don't currently remember how we figured out some of the values used in our process. You can check the manual and compare to ours, but note that *-b* and *-z* should only affect performance, not clustering. However, we set *-e* based on *-z* (following the manual), which does affect results. We need to investigate further.

Table 3. Settings for CROP.
NUMSEQS is the total number of sequences. *BP* is the target sequence length.

-r	5	Clusters of size $leqr$ are considered rare
-s		Specifies that 97% similarity needed to cluster
-b	$NUMSEQS/50$	Number of blocks
-z	$(150000/BP) - 0.1 * (150000/BP)$	Max number of sequence a block can hold
-e	$[-z] * 10$	Number of MCMC iterations

3.3.2 Correct the OTU size

CROP records what sequences end up in an OTU, as well as the total number of sequences represented by an OTU. However, CROP has no knowledge that `obiclean` and `obimerge` already collapsed duplicate/similar sequences into. We call `CROP_size_fix.sh` to get the actual OTU counts.

3.4 Taxonomic Assignment

Taxonomic assignment can be done using either BLAST or VSEARCH. (SAP coming soon). The exact scripts and slurm wrappers depend on method selected.

3.4.1 Run taxonomic assignment program in parallel

Split sequences into chunks and BLAST in parallel. Uses the BLAST ignore list (provided in parameters) to skip assignment to unwanted sequences. For example, we typically try to avoid assignment to environmental sequences.

3.4.2 Concatenate into single FASTA

The small chunks are concatenated into a single FASTA file.

3.4.3 Convert to Charon format

Charon is the name of our own CSV format for assigned sequences. The purpose is that we can write conversion scripts to have assignments from various taxonomic assignment programs into a standard. The Charon file has columns specified by Table 4.

Table 4. Columns for Charon CSV file

Query Sequence ID	Assignment Sequence ID	Scientific Name	NCBI GI	Number of sequences in OTU

3.5 OTUs VS Samples

Formerly: *Critters VS Tubes*.

Will explain soon, but basically just a script that makes a column for each sample and a row for each OTU. The cells have a count of the number of sequences of that OTU in that sample. Also has columns of the scientific names for various phylogenetic levels. Also has extra information columns such as the sequence, BLAST scores, etc. No limit on added extra columns to this CSV file.

4 EXAMPLE RUN

4.1 Prepare Data

4.2 Run Pipeline Script

The following bash script executes `charybdis_generic.sh`, an implementation of the GCL pipeline. The code segment shown is a single line of bash, using \ for clarity to break it into one line per option.

```
bash /work/hobi/GCL/GCL_charybdis/pipelines/charybdis_generic.sh \
    -p Simons-H1_ATTACTCG-TAATCTTA_L001 \
    -i /work/hobi/GCL/20180330-Simons/Simons-H1/in \
    -o /work/hobi/GCL/20180330-Simons/Simons-H1/out \
    -n 20 \
    -b /work/hobi/GCL/db/NCBI_BLAST_DBs/nt \
    -d /work/hobi/GCL/db/GI_lists/environmental.NCBI_nucl__SORTED.gi \
    -c /work/hobi/GCL/db/BOLD_FULL.fasta \
    -x 313 \
    -g /work/hobi/GCL/GCL_charybdis/bin \
    -t /work/hobi/GCL/db/TAXO
```

Each option will be explained, and the first 10 lines of each file will be displayed. This way, you can compare the format of your input files to those used for the example. Do not trust the spacing seen here. Tabs may become spaces, etc. See the official file format specifications.

-p Name of project.

We used `Simons-H1_ATTACTCG-TAATCTTA_L001` to match the names of the input FASTQ files. The important thing is that each project should have a unique name for keeping projects separate/organized.

-i Input directory.

This is where the input files should be placed. A slight abuse of terminology, since `<projectname>.samples.txt` is generated by the pipeline, by reading sample IDs from `<projectname>.barcodes.txt`. Note that the FASTQ sequences were shortened to fit on the page. A sequence of five periods (`.....`) represents a removed segment.

```
[ekrell@hpcm Simons-H1]$ ls in
        Simons-H1_ATTACTCG-TAATCTTA_L001.barcodes.txt
        Simons-H1_ATTACTCG-TAATCTTA_L001_forward.fastq
        Simons-H1_ATTACTCG-TAATCTTA_L001_reverse.fastq
        Simons-H1_ATTACTCG-TAATCTTA_L001.samples.txt

[ekrell@hpcm Simons-H1]$ head in/Simons-H1_ATTACTCG-TAATCTTA_L001_forward.fastq
    @MISEQ01:242:000000000-BLT5R:1:1101:15746:1350 1:N:0:ATTACTCGTAATCTTA
    ACTTGATAGACCTCGGGGTGGCCGAAGAACCAGAATAGGTGTTGGTAAAGAATTGGGTCTCC .....
    +
    CCCCCFFFFFFFGGGGGGGGGGHGGGGGHHHHHHHHHHHHHHGHHGHHHHHHHHHHHHGHHHH .....
    @MISEQ01:242:000000000-BLT5R:1:1101:16137:1376 1:N:0:ATTACTCGTAATCTTA
    ATCACGGGAACTGGATGAACAGTTTATCCTCCCCTTGCCGGCAACCTGGCCCACGCAGGGGC .....
    +
    CDDDDDBCCBCFGGGGGGGGGGGHHHHHHHHHHGGHHHHHGGGGGHHHHHHHHGGGGGGGGGG .....
    @MISEQ01:242:000000000-BLT5R:1:1101:15575:1386 1:N:0:ATTACTCGTAATCTTA
    GTCCGCGGTACTGGATGAACAGTATACCCCCCTTTAGCAGCAGCCATTGCACATGCAGGTGC .....

[ekrell@hpcm Simons-H1]$ head in/Simons-H1_ATTACTCG-TAATCTTA_L001_forward.fastq
    @MISEQ01:242:000000000-BLT5R:1:1101:15746:1350 1:N:0:ATTACTCGTAATCTTA
    ACTTGATAGACCTCGGGGTGGCCGAAGAACCAGAATAGGTGTTGGTAAAGAATTGGGTCTCCCCCA .....
    +
    CCCCCFFFFFFFGGGGGGGGGGHGGGGGHHHHHHHHHHHHHHGHHGHHHHHHHHHHHHGHHHHHGGG .....
    @MISEQ01:242:000000000-BLT5R:1:1101:16137:1376 1:N:0:ATTACTCGTAATCTTA
    ATCACGGGAACTGGATGAACAGTTTATCCTCCCCTTGCCGGCAACCTGGCCCACGCAGGGGCATCA .....
    +
    CDDDDDBCCBCFGGGGGGGGGGGHHHHHHHHHHGGHHHHHGGGGGHHHHHHHHGGGGGGGGGGGHHH .....
    @MISEQ01:242:000000000-BLT5R:1:1101:15575:1386 1:N:0:ATTACTCGTAATCTTA
    GTCCGCGGTACTGGATGAACAGTATACCCCCCTTTAGCAGCAGCCATTGCACATGCAGGTGCATCT .....

[ekrell@hpcm Simons-H1]$ head in/Simons-H1_ATTACTCG-TAATCTTA_L001.barcodes.txt -n 5
    #exp         sample   tags            forward_primer      reverse_primer
    Simons-H1    1901-B   ACAGTG:CCGTCC   GGWACWGGWTGA .....  TANACYTCNGGRTGN .....
    Simons-H1    1685-C   ACAGTG:GTTTCG   GGWACWGGWTGA .....  TANACYTCNGGRTGN .....
    Simons-H1    1759-B   ACTGAT:ATGTCA   GGWACWGGWTGA .....  TANACYTCNGGRTGN .....
    Simons-H1    1774-X   ACTGAT:GCCAAT   GGWACWGGWTGA .....  TANACYTCNGGRTGN .....

[ekrell@hpcm Simons-H1]$ head in/Simons-H1_ATTACTCG-TAATCTTA_L001.sampledescs.csv
-n5
    Tube,Description
    01-A,Larimus_fasciatus
    05-B,Larimus_fasciatus
    05-C,Larimus_fasciatus
    08-A,Larimus_fasciatus
```

-o Output directory.
Files generated by the pipeline, including numerous intermediate file, are placed here.
-n Number of parallel instances.
The nodes on TAMUCC's HPC cave 20 cores, so 20 were used.
-b BLAST Database.
Specify path to BLAST database. We are using the nucleotide (nt) database,
which is downloaded from `ftp://ftp.ncbi.nlm.nih.gov/blast/db/`.
-d List of GIs for BLAST to ignore This is a list of GIs, where each GI specifies a sequence in the BLAST database to ignore. This file is mandatory, but may be blank (See Section 5.2).
We attempt to remove environmental DNA, so our list has GIs of environmental sequences.

```
[ekrell@hpcm Simons-H1]$ head -n 5 ...../environmental.NCBI_nucl__SORTED.gi
  1000014437
  1000014439
  1000014441
  1000014443
  1000014445
```

-c Chimera Database. We use a FASTA file generated from the BOLD database. But, we ended up doing *de novo* chimera detection. Currently this file is required and just not being used (See section 5.3).
-x Target sequence length (basepairs). This number is based on the primers used during PCR.
-g Directory of charybdis scripts.

```
ls /work/hobi/GCL/GCL_charybdis/bin
  \                                   fastq-splitter.pl
```

```
AddBlast.slurm                        filterReads.slurm
AddVsearch.slurm                      mergeReads.sh
AssignToMarkers.sh                    mergeReads.slurm
AssignToMarkers.slurm                 ObiToolsPipeline.sh
blast_10custom_to_charon_OTU.R        ObiToolsPipeline.slurm
BlastMeta.sh                          ObiToolsPipeline_stats.sh
BlastMeta.slurm                       OTU_CVT_addBlast.R
ClusterOTU.sh                         OTUvsTube.slurm
ClusterOTU.slurm                      split_by_marker.R
CombineAndCROP.sh                     vsearch_blast6custom_to_charon-BLASTDB_OTU.R
crittersVStubes_OTU.R                 vsearch_blast6custom_to_charon_OTU.R
CROP_size_fix.sh                      vsearch_getTAXIDfromBOLDseqid.sh
CROP.slurm                            Vsearch.sh
determine_marker.R                    Vsearch.slurm
determine_marker.sh
```

-t Directory with NCBI taxonomy database Taxonomy database downloaded from `ftp://ftp.ncbi.nlm.nih.gov/pub/taxonomy/`.

-g Directory of charybdis scripts.

```
[ekrell@hpcm pipelines]$ ls /work/hobi/GCL/db/TAXO
        Accesion2Taxid   delnodes.dmp   gc.prt         merged.dmp   nodes.dmp    taxdump.tar.g
        citations.dmp    division.dmp   gencode.dmp    names.dmp    readme.txt
```

5 AREAS FOR IMPROVEMENT

5.1 Support NCBI Accession IDs

NCBI is phasing out GIs (sequence IDs), which this pipeline uses at various steps (NCBI, 2016). Accession IDs are the current standard.

5.2 Pipeline should not require BLAST ignore list

Currently this is required. If you don't need it, you can supply a blank file. Not very elegant. Would be simple to make it optional.

5.3 Chimera database should not be mandatory

The generic pipeline uses *de novo* chimera detection, but still requires the chimera database. A better solution would be to default to *de novo*, but use the database if supplied.

6 STATISTICAL ASSIGNMENT PACKAGE (SAP)

We have SAP working, but not meshed into the pipeline yet. So it has its own section, at least for now.

SAP (Munch et al., 2008) differs from typical assignment software (BLAST, VSEARCH) in that it is based on computing a posterior probability that the query sequence is a member of a particular clade. Two major phases are involved in an SAP taxonomic assignment. First, BLAST is executed to find a set of similar sequences. Since similar sequences should indicate similar biological properties, the set of accepted BLAST sequences are called the set of homologues with the query. To strengthen the assumption that the sequences are homologous, there is a limit on how distant the furthest apart homologues can be. Several parameters are used to control BLAST's sequence search and also to control how SAP chooses which of those sequences to include in the homologue set. It can and does happen that a single species has several sequences in the database. However, a particular taxonomic group appears only once in the phylogenetic tree. Therefore, SAP focuses on diversity and only uses the best match for a given species. While this is necessary to construct meaningful trees, it also means that the species which only appears once in a set of BLAST results is given equal weight to a predominate species result. The set of homologues and the query sequence are used to sample a large number of trees using Markov Chain Monte Carlo. The posterior probability for any taxonomic group is based on the proportion of trees in which the query sequence was in that group. A simple case that shows how this can differ from BLAST occurs when several species from the same genus are present in the homologue set. Where BLAST would simply list the hits in order of sequence ID, SAP determines that it cannot reliably be determined what species the query belongs to. A top hit may differ little from the hits just below it, and it is essentially arbitrary to assume that the top hit is the exact species when it scored closely against other species as well. The SAP result would give a low posterior probability to any particular sequence, but the genus would have a very high probability.

The above text taken from GCL's work-in-progress manuscript "An Evaluation of Software for Taxonomic Assignment with DNA Metabarcoding"

6.1 Fixing SAP

6.2 Running SAP

6.3 Evaluating SAP

6.3.1 Categorize SAP results

6.3.2 Generate SAP report

6.4 SAP outstanding issues

6.5 Complete SAP example

REFERENCES

Hao, X., Jiang, R., and Chen, T. (2011). Clustering 16s rrna for otu prediction: a method of unsupervised bayesian clustering. *v*, 27:611–8.

Munch, K., Boomsma, W., Huelsenbeck, J. P., Willerslev, E., and Nielsen, R. (2008). Statistical assignment of dna sequences using bayesian phylogenetics. *Systematic biology*, 57(5):750–757.

NCBI (2016). Ncbi is phasing out sequence gis – here's what you need to know.

Ratnasingham, S. and Hebert, P. D. (2007). bold: The Barcode of Life Data System (http://www.barcodinglife.org). *Mol. Ecol. Notes*, 7(3):355–364.

Rognes, T., Flouri, T., Nichols, B., Quince, C., and Mahe, F. (2016). VSEARCH: a versatile open source tool for metagenomics. *PeerJ*, 4:e2584.

TingChenLab (2017). Crop. https://github.com/tingchenlab/CROP.